

---

**EIE131 Digital Circuits Laboratory**

**PLD Experiment –**

**Software and Hardware  
Introduction**

# Experiment Schedule

Index	Topic	Hours	Teaching Method
1	Introduction to PLD Design and Simple Practice	6	Lecture
2	PLD Drill Practice	1+5(3)	Lecture+Lab
3	Experiment I : <b>4-bit Full Adder Design</b>	1+5(3)	Lecture+Lab
4	Experiment II : <b>Sequence Generator Design</b>	1+5(3)	Lecture+Lab
5	Review	1+2	Lecture+Lab
6	Mid-term Exam.	3	Lab
7	Experiment III : <b>4-bit Serial Adder Design</b>	1+5	Lecture+Lab
8	Experiment III : <b>4-bit Serial Adder Design</b>	3	Lab
9	Review	1+2	Lecture+Lab
10	Final Exam.	3	Lab

# Reference Materials and Assessment Approach

---

## *Reference Materials*

1. PLD Introduction PPT
2. ispLEVER Learning Video
  - a. isp01\_SchematicInput(ConstructComponent)
  - b. isp02\_SimulateWTestVector(BUS\_waveform)
3. PLD Exercises and ABEL materials
4. Docs and PPTs for each Experiment

## *Assessment Approach*

**Exercise:** 10% ; **Experiment:** 30% ;  
**Mid-term Exam:** 30% ; **Final Exam:** 30%

# Objective

---

- Train the student's **practical, operation and problem-solving abilities** for digital circuit design.
- **Finish** each experiment by yourself **individually**.
- **Preparing** the each experiment before the lab.
- The experimental **results will be checked** in class and **performance will be recorded**.
- **Write the report** after each experiments and **hand in on time**.

# Marking standard (full mark = 10)

---

- 0 :** To be absent from the Lab without receiving the permit for the leave.
- 3 :** **Attend the Lab** but cannot finish all the requirements and the report is a very poor or copy version.
- 5 :** Attend the Lab and **finish all the requirements** but the report is a poor or partly-copy version.
- 6.5:** Attend the Lab and finish all the requirements and write the report individually **but it is not standard.**
- 8 :** Attend the Lab and finish all the requirements and write the report individually and it **does include a good summary and tell what you have learned.**
- 10 :** Finish all above requirement **by using other different methods or can also finish the drill problem** after the standard experiment requirement.

# Experiment Report Standard

---

- Experiment Requirement, Experiment Target,
- Experiment Steps
- Experiment Content
  - Circuit Diagram (\*.sch)
  - Circuit Design in Abel (if any, \*.abl)
  - Circuit Test Vector (\*.abv)
  - Circuit Function Simulation Results including the critical waveforms
- Analysis and Summary
- State what you have learned from the experiment.

# Lab Grouping

---

**This Class are held at the following time:**

- **Friday afternoon from 15:30 to 18:20**
- **Students should attend the Lab *on time* because the experimental equipment is enough for extra ones.**
- \* **You should copy the circuit diagram, test vector and output waveforms for each experiment so that you can write the report easily.**

---

# **Digital Circuit Design**

## **PLD Experiment –**

### **Introduction to In-System Programmable Device (ISP)**



# Introduction to In-System Programmable Device (ISP)

- Introduction to **Programmable Logic Device (PLD)**
- The **Structure** of In-System Programmable Device (**ispLSI1032E**)
- **Programming Port and Programming Process** of ispLSI device
- Introduction to **ispLEVER Project Navigator**
- Introduction to **PLD Experimental System (PLD-PAC-1)**

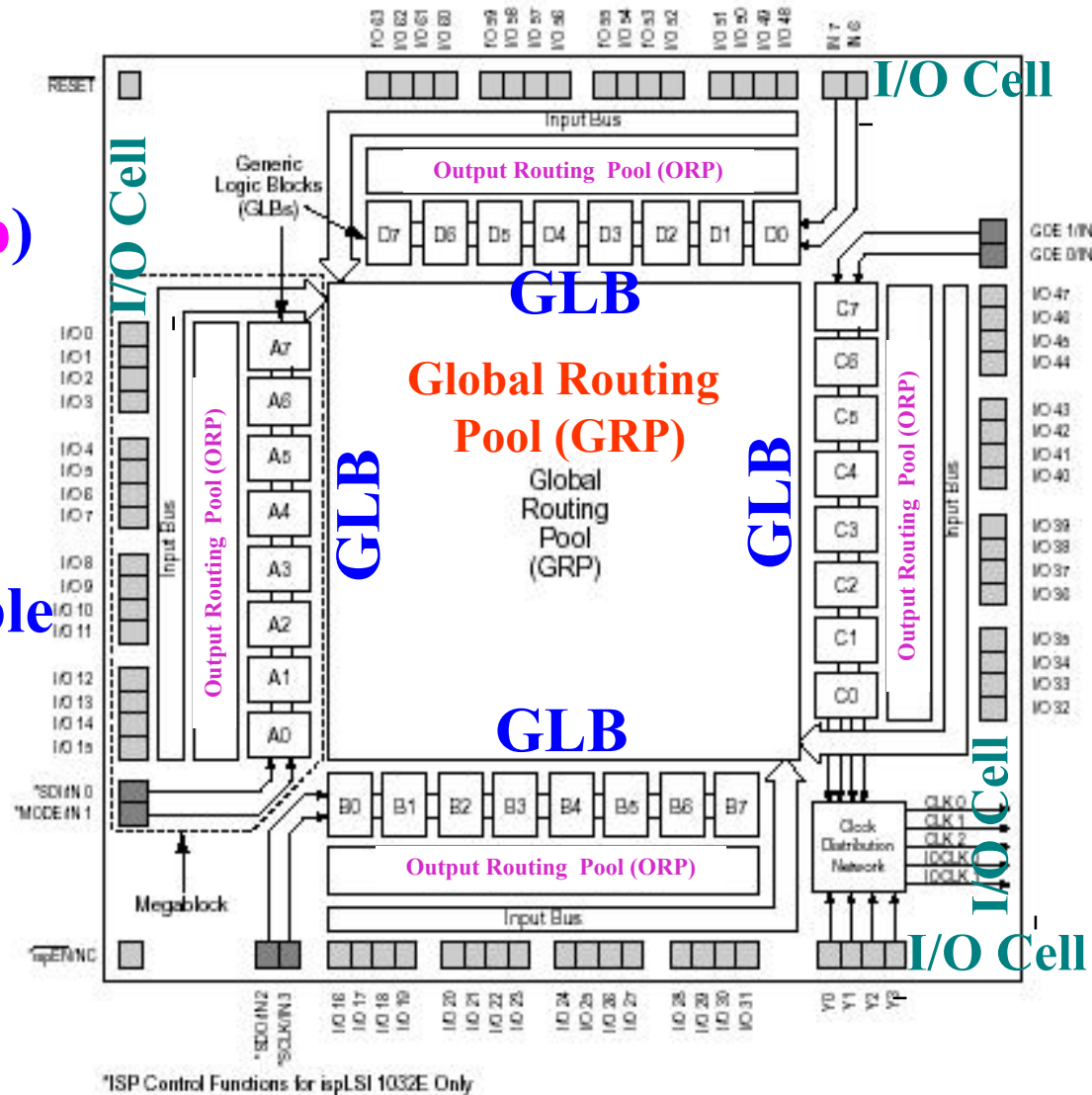
# Introduction to Programmable Logic Device (PLD)

---

- The drawbacks of **traditional hardware design** method are: difficult to modify, long design period, high cost and low reliability.
- **ASIC** (Application Specific Integrated Circuit) is **high** reliability, high speed, low power consumption and small size **but long design period and high cost**.
- **PLD** (Programmable Logic Device) appears **as a balance**
  - **PLD** (Programmable Logic Device):GAL (Generic Array Logic), PAL.
  - **CPLD** (Complex Programmable Logic Device)
  - **FPGA** (Field Programmable Array Logic)
  - **ISP** (In System Programmable) **PLD**  
**ispLSI1032E(70LJ84)** from Lattice is the **main PLD in the Experimental System**.

# The Structure of In-System Programmable Device (ispLSI1032E)

- **6000 Gates**
- **64 General I/O pins**
- **192 Registers (D flipflop)**
- **Highest working frequency = 125MHz**
- **I/O Voltage level is TTL**
- **In-System Programmable**
- **4 Clock Input**
- **2 Output Enabling Control**



# The Structure of In-System Programmable Device (ispLSI1032E)

---

- **GLB (Generic Logic Block)**

**Totally 32 blocks A0~A7, B0~B7, C0~C7, D0~D7 are the basic logic units for ispLSI1032E. For each GLB, there are**

- \* 18 inputs** (two are used for special purpose)

- \* programmable AND array** which forms 20 Product terms

- \* 4 Output Logic Macro Cell** can be configured to **combinational output (.com) or register output (.reg)**

- **I/O Cell (Input/Output Cell)**

**64 general I/O pins.** When it is Input, I/O cell will send the signal to GRP, when it is Output, I/O cell sends the signal from ORP to the pin.

# The Structure of In-System Programmable Device (ispLSI1032E)

---

- **Global Routing Pool (GRP)**

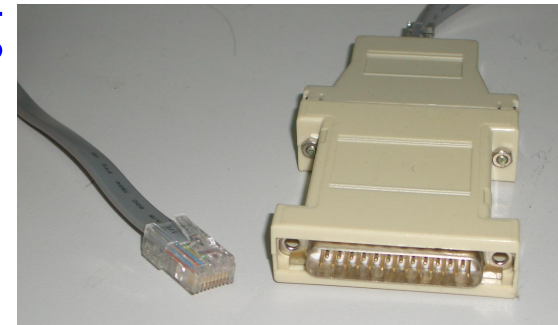
Provide the programmable connections **between GLBs**. The signal delay through GRPs are fixed and depends on the number of GLB passing and can be expected.

- **Output Routing Pool (ORP)**

Provide the connections between GLB and I/O cell thus increasing the flexibility of I/O.

# Programming Port and Programming Process of ispLSI device

- ❑ Programming Port of ispLSI1032E includes five pins as follows: **ispEN**、**SDI/IN0**、**MODE/IN1**、**SDO/IN2** and **SCLK/IN3**.
- ❑ **ispEN** is the programming enable and it is **Low**, the other pins are **SDI**、**MODE**、**SDO** and **CLK**. When it is **High**, they become specific inputs: **IN0**、**IN1**、**IN2** and **IN3**.
- ❑ Programming Port of ispLSI1032E is connected with Parallel Port of PC. When ispEN is low, all I/O go to Hi-Z and the chip enters the programming state.
- ❑ Download is a process of programming and can be done after connecting the programming port with the parallel port of PC



# Introduction to ispLEVER Project Navigator

---

- ispLEVER Project Navigator (shortly the ispLEVER) can be used to develop the chips in ispLSI1000、 2000、 3000、 5000、 6000、 8000 series
- The chip used on the Experimental Box is ispLSI1032E-70LJ84 which should be selected while building a new project in ispLEVER
- ispLEVER supports Top-Down Design and Down-Top Design flexibly
- ispLEVER has full functions and can finish the whole process including Design Input, Compile, Simulation, Analysis and Download

# Introduction to ispLEVER Project Navigator

---

□ ispLEVER has several Methods to describe the Design and they can be used mixedly and feasibly.

\* **Logic Diagram Input**—Top Design is used mostly

\* **ABEL Input**—**Logic Equation, Truth Table, State Diagram** (Select according to the actual situation)

\* **VHDL** (will be discussed in year 3)

□ With **Waveforms Display and Editing** functions thus debugging more directly and conveniently

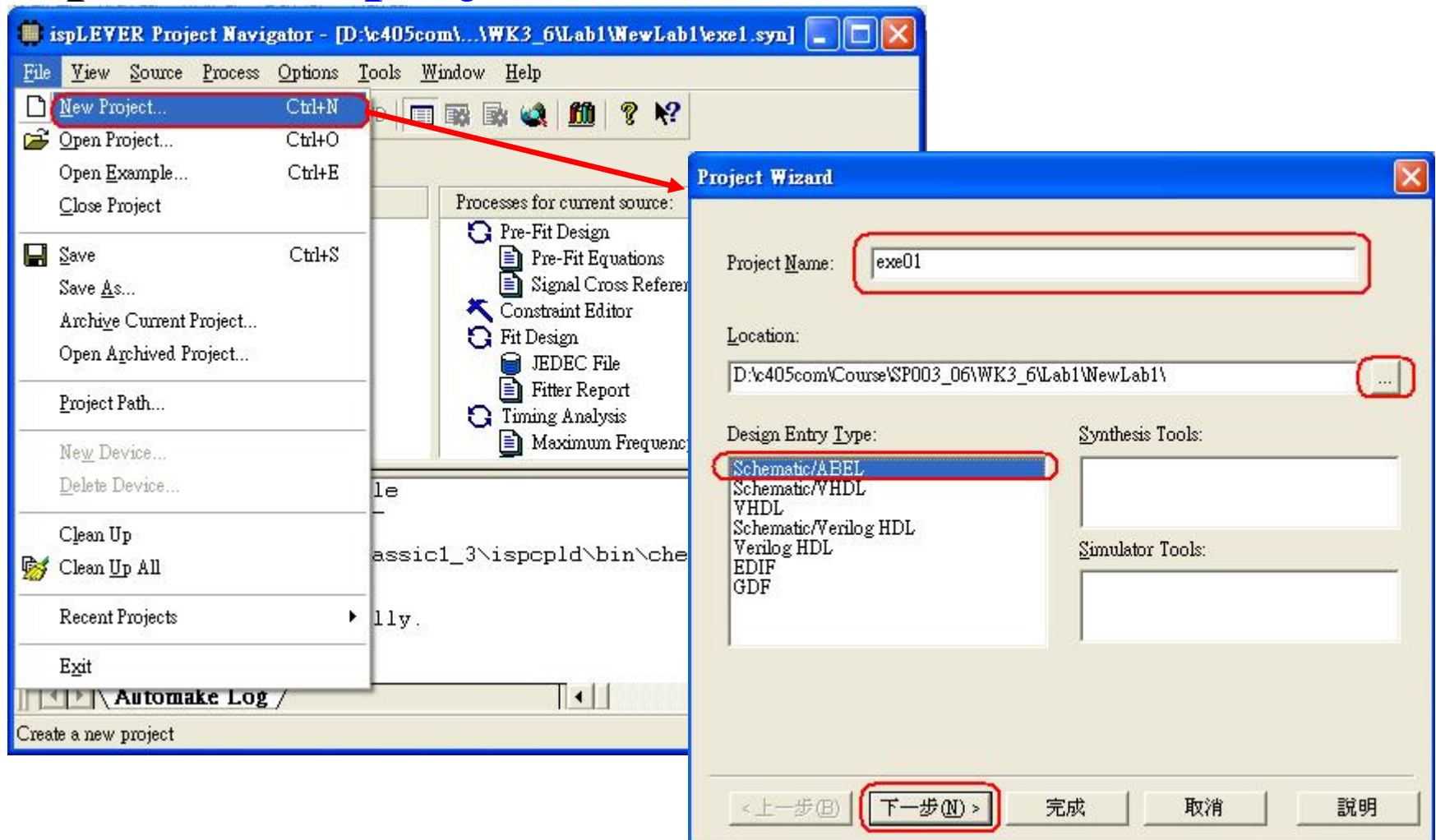
□ **File Extension** in ispLEVER:

Project File(\*.**SYN**), Circuit Diagram File(\*.**SCH**),  
Test Vector File (\*.**ABV**), ABEL Input File(\*.**ABL**),  
**Waveform Input File(\*.WDL)**



# New Project in ispLEVER Project Navigator

- Open a new project and select Schematic/ABEL



# Select ispLSI1032E-70LJ84 as the Device

**Check** the **Obsolete Devices** → Select **ispLSI 1K**  
**Devices** → **ispLSI1032E** → **Speed Grade = 70MHz**

**Project Wizard - Select Device**

Select Device:

Family: **ispLSI 1K Device**

Device: **ispLSI1032E**

Speed grade: (MHz) **70**

Package type: **84PLCC**

Operating conditions: **Commercial**

Part Name: **ispLSI1032E-70LJ84**

☐ Use I/O Assistant Flow ☒ Show Obsolete Devices

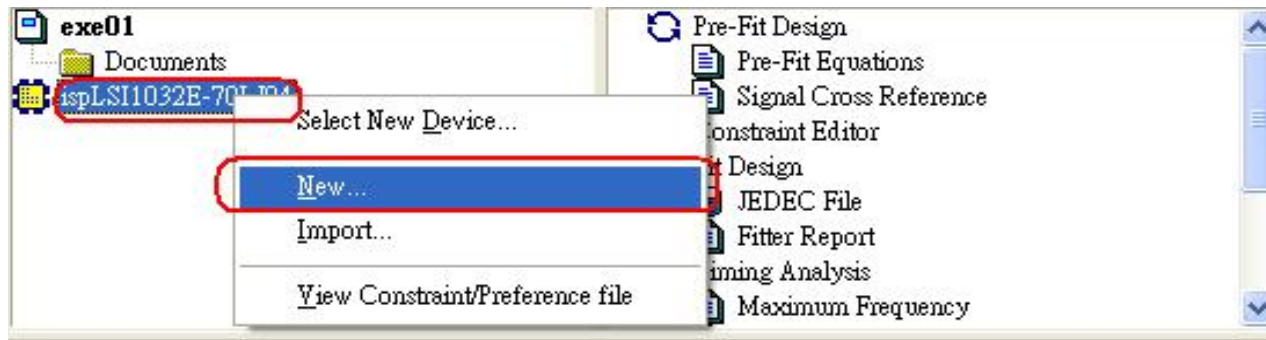
< 上一步(B) **下一步(N) >** 完成 取消 說明

**Device Information:**

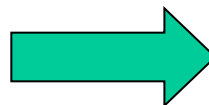
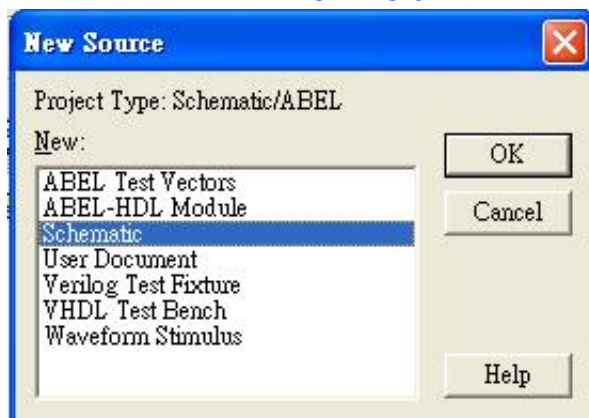
Status:	Obsolete
Density:	6000
Logic cells:	128
I/O cells:	64
I/O pins:	64
Dedicated input:	8
Output enable:	2
Icc:	190 mA

# New Schematic

- Right click the **ispLSI1032E-70LJ84** → **New** → **Schematic** → Enter the Circuit Name

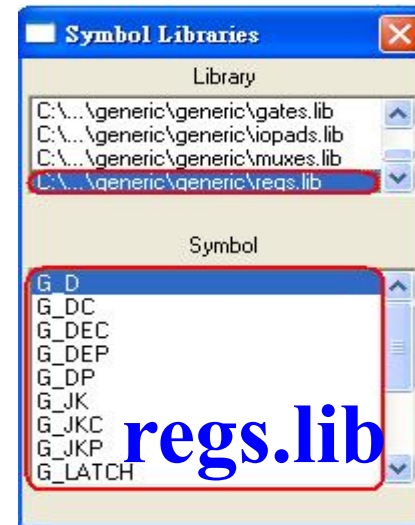
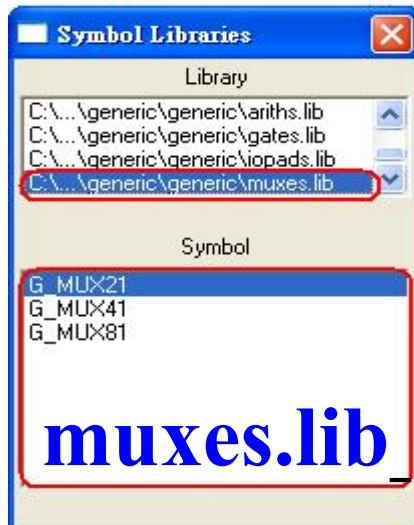
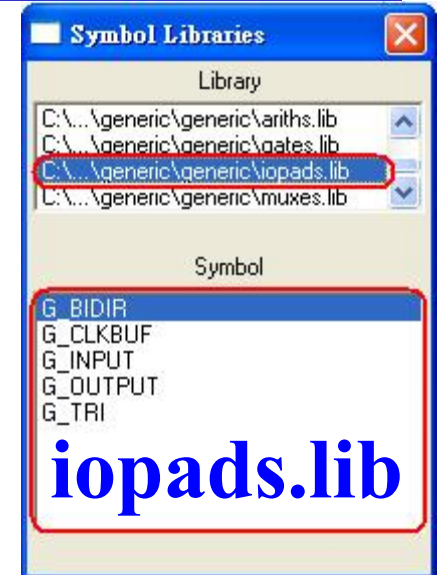
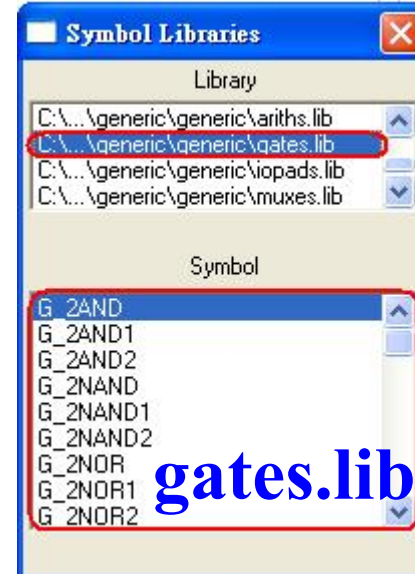
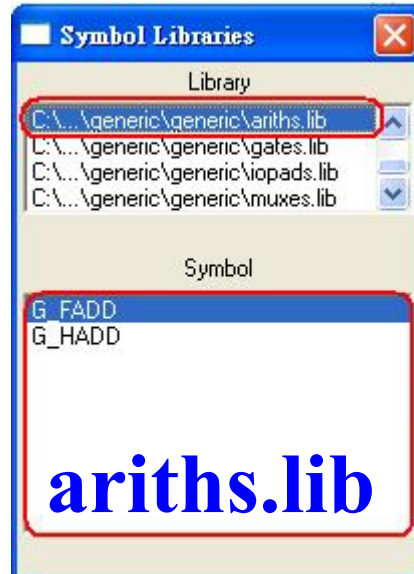


Select **ABEL Test Vectors** to input the test-vectors  
Select the **ABEL-HDL Module** to design the circuit in ABEL-HDL



# Five Main Symbol Libraries

- Add Symbol in Library:


















ariths.lib  
gates.lib  
iopads.lib  
muxes.lib  
regs.lib



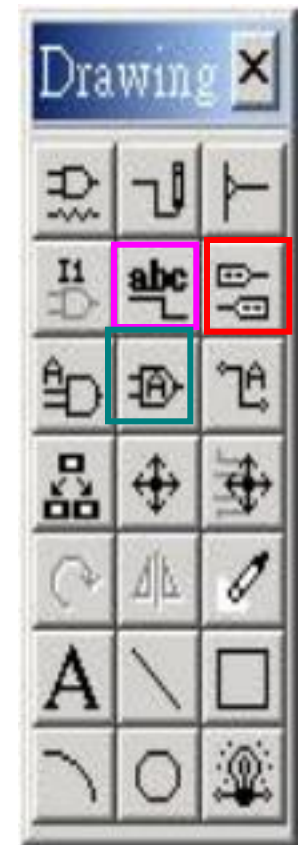
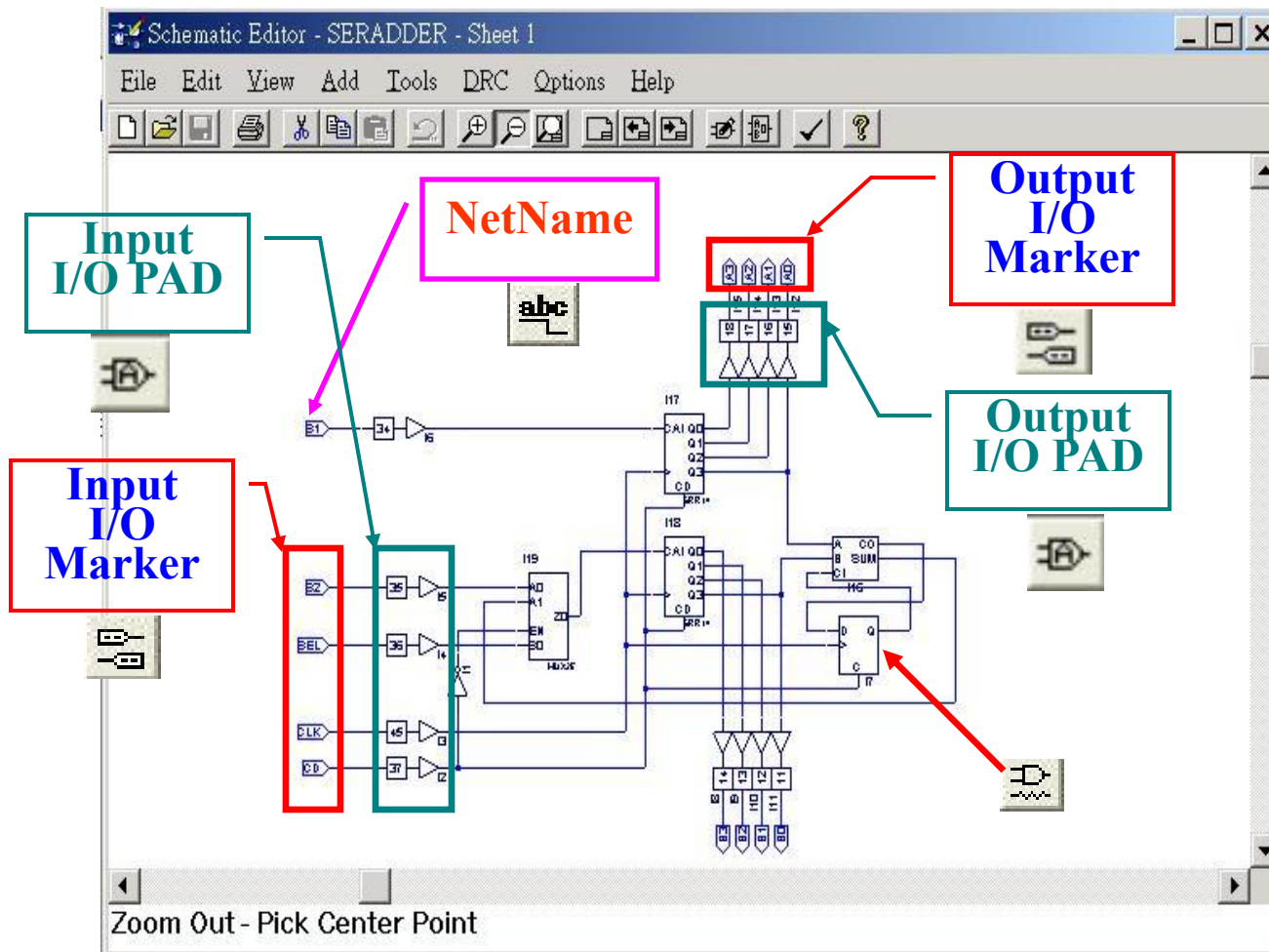
# Drawing Toolbar



		
Add Symbol	Add Wire	Add Bus Tap
		
Add Instant Name	Add Net Name	Add I/O Marker
		
Edit Pin Attribute	Edit Symbol Attribute	Edit Net Attribute
		
Duplicate	Move	Drag
		
Rotate	Mirror	Delete
		
Draw Text	Draw Line	Draw Rectangle
		
Draw Arc	Draw Circle	Highlight

# Logic Diagram Demo

## ■ Logic Diagram Input (\*.sch)



# Test Vector & Functional Simulation

- Functional Simulation** according to **Test Vector(\*.abv)**  
One project can only has one vector file

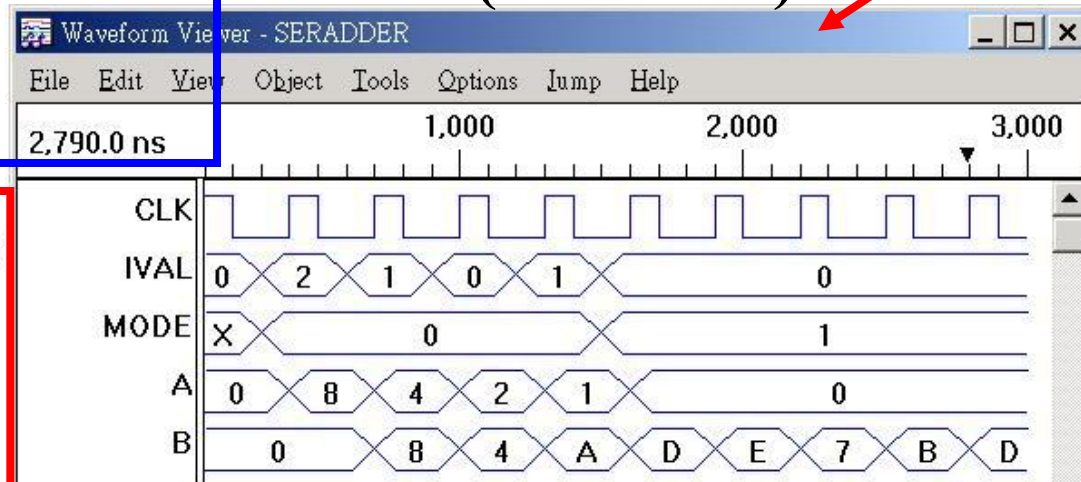
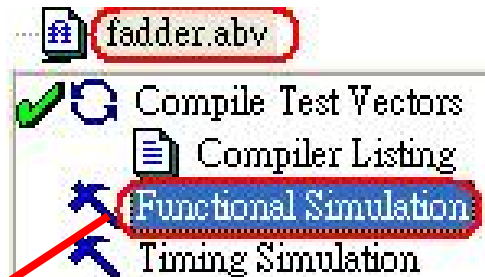
```
seradder.abv
module seradder
A3..A0,B3..B0,S1,S2,SEL,CLK,CD PIN;
A=[A3..A0];
B=[B3..B0];
C=.C. ;
X=.X. ;
IVAL = [S1,S2];
MODE = [SEL,CD] ;
CLEAR = [X,1];
PUSH = [0,0];
ADD = [1,0] ;
```

```
TEST_VECTORS
([IVAL,MODE,CLK]->[A,B])
[0,CLEAR,C]->[X,X];

[2,PUSH,C]->[X,X];
[1,PUSH,C]->[X,X];
[0,PUSH,C]->[X,X];
[1,PUSH,C]->[X,X];

[0,ADD,C]->[X,X];
[0,ADD,C]->[X,X];
[0,ADD,C]->[X,X];
[0,ADD,C]->[X,X];
[0,ADD,C]->[X,X];
END
```

A3..A0 **pin** ;  
A=[A3..A0];  
C=.C. (Clock)  
X=.X. (don't care)



**TEST\_VECTORS** ([input list]->[output list])  
e.g. **TEST\_VECTORS**  
([IVAL,MODE,CLK]->[A,B])  
[0,CLEAR,CLK]->[X,X];

# ABEL-HDL

(ABEL-Hardware Description Language)

---

- ABEL-HDL is one **kind of logic description language** and the most popular one in designing PLD. ABEL-HDL source code is **written in ASCII format in a text file.**
- **Most of EDA** (Electronics Design Automation) tools **support ABEL-HDL** design method
- ABEL-HDL is **very easy to learn** for the **beginner** of PLD design.



# Key words in ABEL

---

CASE	FUSES	PIN
DEVICE	GOTO	STATE
ELSE	IF	STATE_DIAGRAM
ENABLE	IN	TEST_VECTORS
END	ISTYPE	THEN
ENDCASE	LIBRARY	TITLE
ENDWITH	MACRO	TRUTH_TABLE
EQUATIONS	MODULE	WHEN
FLAG	NODE	WITH

**Note:**

**Keywords are not case sensitive and *cannot* be used to name the component, pin, node, collection, macro and signal.**

# Data Representation and Special Constant in ABEL

---

Radix(Base)	Format	Example	Decimal value
2	<b>^b</b>	<b>^b101</b>	<b>4</b>
8	<b>^o</b>	<b>^o77</b>	<b>63</b>
10	<b>^d</b>	<b>^d43 or 43</b>	<b>43</b>
16	<b>^h</b>	<b>^h0F</b>	<b>15</b>

<b>.C.</b>	<b>Positive-Pulse Clock Input</b>	<b>.K.</b>	<b>Negative-Pulse Clock Input</b>
<b>.U.</b>	<b>Clock rising edge</b>	<b>.D.</b>	<b>Clock rising edge</b>
<b>.F.</b>	<b>The float Input/Output</b>	<b>.Z.</b>	<b>Hi-Z Input/Output</b>
<b>.X.</b>	<b>Don't Care</b>	<b>.P.</b>	<b>Register preset</b>

# Three Operations in ABEL

Type	Operator	Operation	Example
Arithmetic	- << >> * / % + -	Negative Left Shift Right Shift Multiplication Unsigned Division Division Residue Addition Subtraction	-A A<<B(A left-shifts B bits) A>>B(A right-shifts B bits) A*B A/B A%B A+B A-B
Logical	! & # \$ !\$	NOT AND OR XOR XNOR	!A A&B A#B A\$B A!\$B
Relationship	== != < <= > >=	Equal Not Equal Less than Less than or equal to Greater than Greater than or equal to	A==B A!=B A<B A<=B A>B A>=B

# Dot Extension of 'reg' in ABEL

Dot Extension of 'reg'	Function
.AP	Set Input of Asynchronous Register
.AR	Re-set Input of Asynchronous register
.CE	Clock Enable
.CLK	Clock Input of Edge-triggered Flip-Flop
.PTCCLK	Clock Input of Product Term
.D	Input of D Flip-Flop
.FC	Trigger Mode Control Pin
.J	J Input of JK Flip-Flop
.K	K Input of JK Flip-flop
.LD	Pre-load of Register
.LE	Latch Enable (Active Low)
.OE	Output Enable
.PIN	Pin Feedback
.RP	Preset Pin
.Q	Feedback of Register
.R	R Input of RS Flip-Flop
.RE	Reset of Register
.S	S Input of RS Flip-Flop
.SP	Synchronous Preset
.SR	Synchronous Re-set
.T	Input of T flip=flop
.FB	Feedback Signal

```

MODULE test
d7..d0 pin 68..75 istype 'reg';
clk pin 45;
reset pin 34;
D=[d7..d0];
Equation
D.clk = clk ;
D:=D.fb+1;
D.ar=!reset;
END
    
```

# ABEL Input Demo

## ■ ABEL Input (\*.abl)

```
unicnt8.abl
MODULE uncnt
  X,C,Z=.X...C...Z.;
  d7..d0 pin 68..75;
  clk pin 84; "or 84
  rst pin 38;
  cnten pin 39;
  ld pin 40;
  u_d pin 41;
  q7..q0 pin 18..11 istype 'reg';
  data = [d7..d0];
  count = [q7..q0];
  mode = [rst,cnten,ld,u_d];
  load = (mode== [ 0, X, 1, X ]);
  hold = (mode== [ 0, 0, 0, X ]);
  up = (mode== [ 0, 1, 0, 1 ]);
  down = (mode== [ 0, 1, 0, 0 ]);
  reset= (mode== [ 1, X, X, X ]);

EQUATIONS
  when reset then count := 0
  else when load then count := data
  else when up then count := count+1
  else when down then count := count-1
  else when hold then count := count ;
  count.clk = clk ;

TEST_VECTORS ([data,clk, mode] -> count)
  [ X, X, 8 ] -> 0 ;
  [ 5, C, 2 ] -> 5 ;
  [ 5, C, 5 ] -> 6 ;
  [ 5, C, 5 ] -> 7 ;
  [ 5, C, 5 ] -> 8 ;
  [ 5, C, 4 ] -> 7 ;
  [ 5, C, 4 ] -> 6 ;

END
```

- **X=.X.** (don't care)
- **C=.C.** (Clock)
- **Z=Z.** (Hi-Z)
- **d7..d0 pin 68..75 istype 'reg';**  
**/d7..d0 pin 68..75 istype 'com';**
- **data =[d7..d0];**
- **when *statement* then { }**  
**else when *statement* then { }**

## TEST\_VECTORS

**([input list]->[output list])**  
**([data, clk, mode]->[count])**  
**[5,C,0]->[X]**

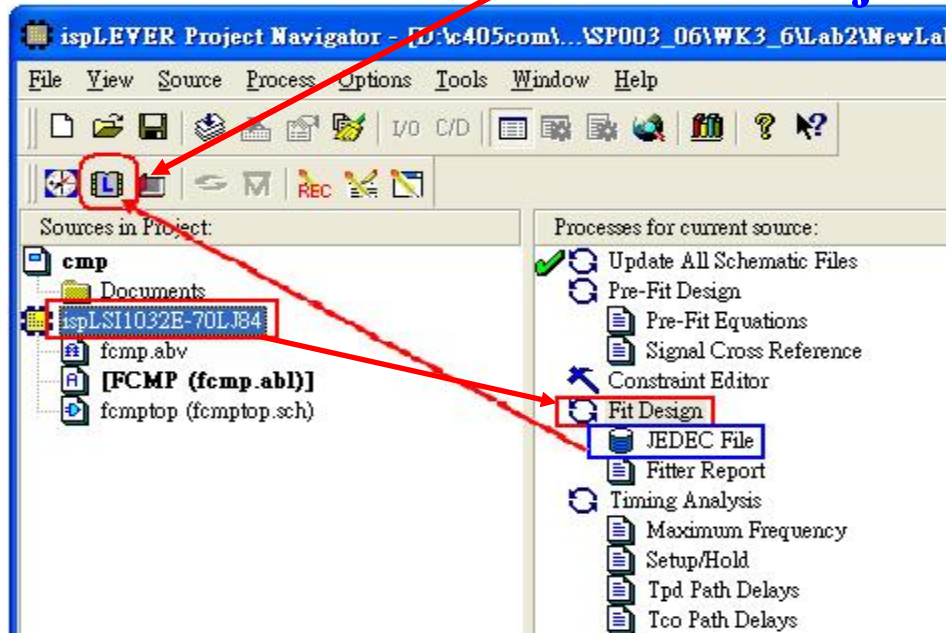
# Download the whole Design to Chip

## 1. Generate the JEDEC File

After finishing the Design Input and Function Simulation, select the Icon of **ispLSI1032E-70LJ84** and click **FIT Design** to generate the **JEDEC File (\*.jed)**.

## 2 Start the ispVM System

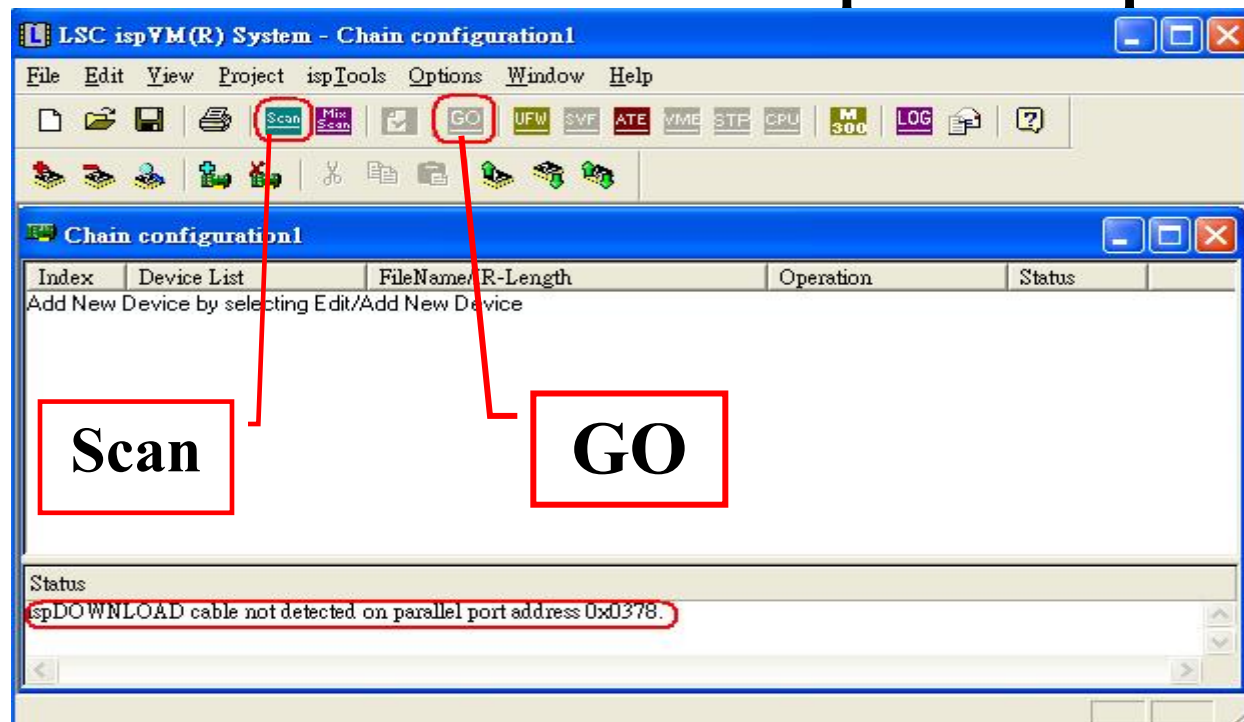
Before downloading, make sure to connect the **download cable** between PC and Experimental Box and **switch on its power**. Click the Tool Icon **ispVM System** to start the **ispVM System** which can **finish the download job**.



# Download the whole Design to Chip

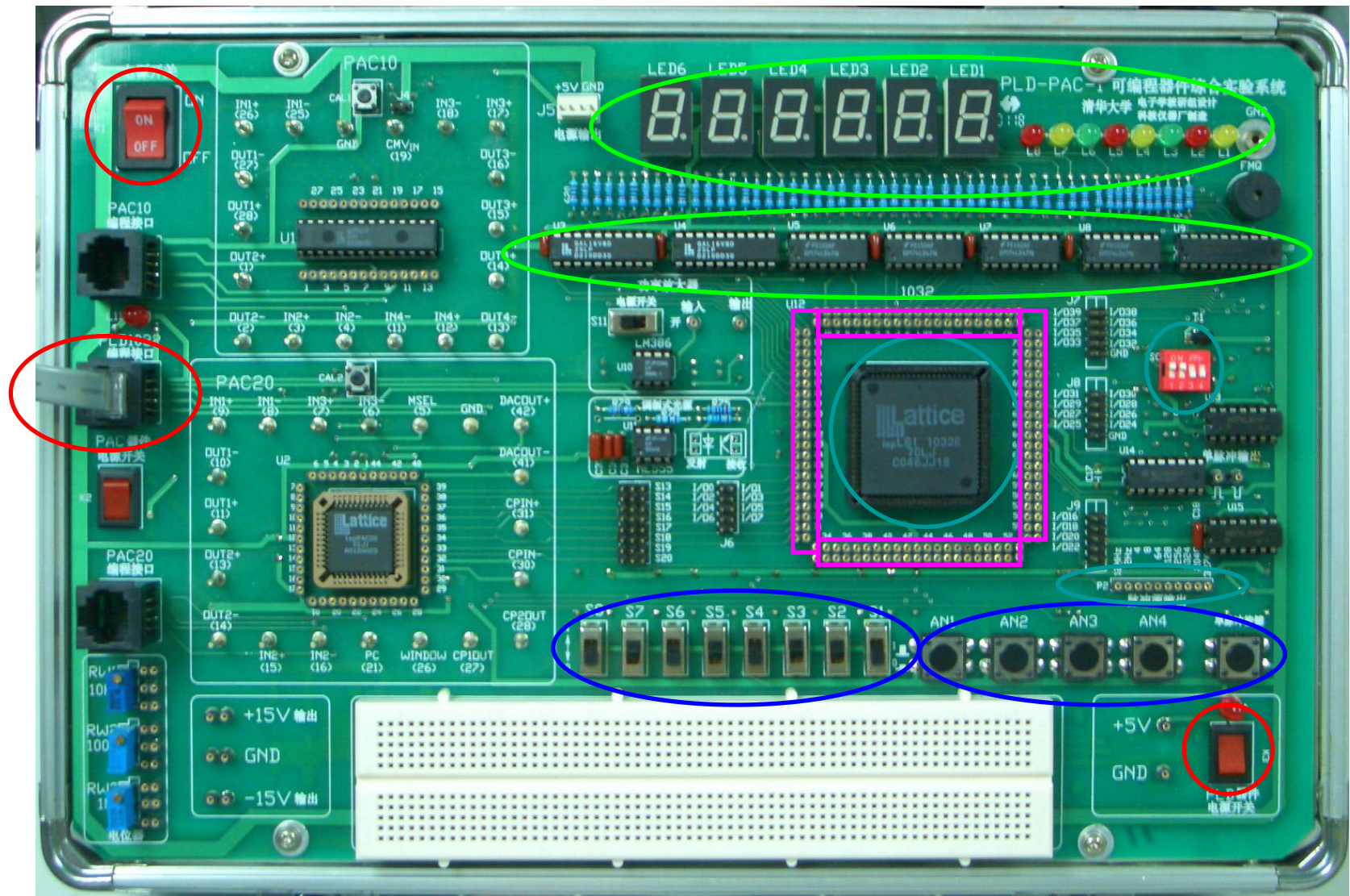
## 3. Scan the download cable connection

After entering the **ispVM System**, click the **ICON Scan** to see if the download cable connection is ok or not. If yes, a **Browse Button** will appear so that you can select the **JEDEC File (\*.jed)**. Select the **JEDEC file** generated and then click the **ICON GO** to download the whole circuit into the chip in the experimental box.



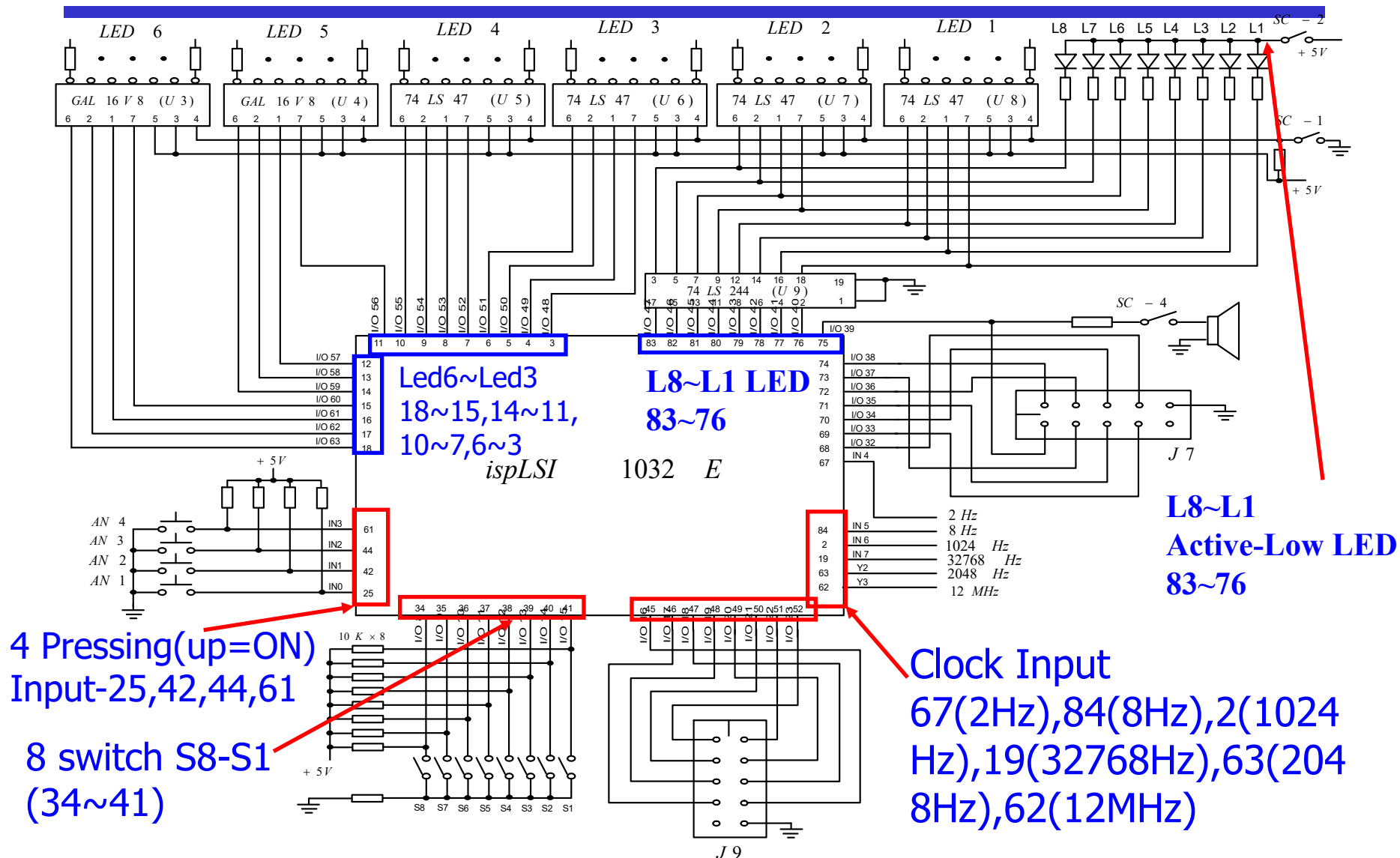


# PLD Experimental System (PLD-PAC-1)



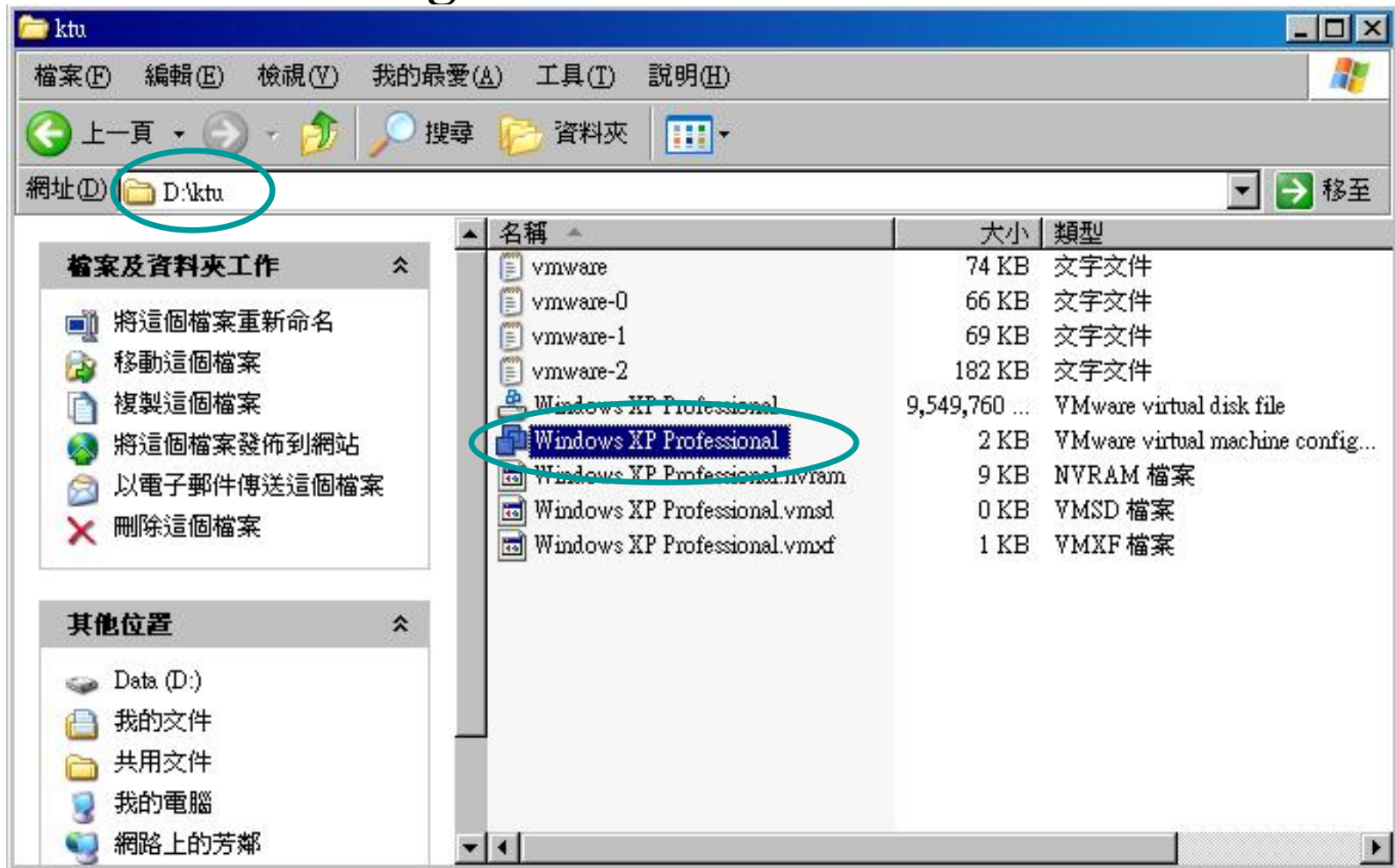


# Internal Circuit of PLD Experimental System(PLD-PAC-1)



# Some Important Things

1. **Experimental Environment** is constructed in a **WM PC** whose **WM image** is in **d:\ktu\** of each PC



# Some Important Things

---

2. Learn the ispLEVER according to the **video and PPTs**
3. In each VM PC of the Lab, create a **PXXX folder in D:\student**(XXX is your last no of your student no.)
4. **Create a new folder inside PXXX for each exercise and experiment** to store all the files in the project.
5. **Backup your files after each Lab** using your U-disk or **email the compressed file** to your email account as a backup.
6. **Folder and file Name should not be beginning with number or in Chinese and cannot use blank and Chinese or excess 8 characters.**
7. The corresponding the video and materials may be found in the path  
**d:\refinfo\CE003\LearnISPByVideo of each VM PC**

# **To be familiar with ispLEVER**

**From d:\RefInfo\CE003\LearnISPByVideo or wechat group, learn the design step according to the given video as follows:**

- a. isp01\_SchematicInput(ConstructComponent**
- b. isp02\_SimulateWTestVector(BUS\_waveform)**

# Installation guide for ispExpert

---

Please download the Lab Software from **wechat** group.

After unzipping the file, follow the steps to install it:

1. Run **classic\_1\_3.exe** by using the administrator right
2. After finishing the installation, copy **license.dat** to the folder **c:\ispLEVER\_Classic1\_3\license**
3. Restart the computer
4. Run the "**ispLEVER Classic Project Navigator**" with administrator right by using the **administrator right**.

# How to build a **BUS** in function simulation

Waveform Viewer - FBREG

File Edit View Object Tools Option

Undo F9  
Redo Shift+F9  
Duplicate F6  
Show...  
Hide F5  
Expand Bus  
Add To Bus...

Show Waveforms

Net Name: All Selected Nets (1)

Show Bus <<

Instances: D

Nets: CLK  
I  
Q0  
Q1  
Q2  
Q3 (2)

Bus Name: Q (5)

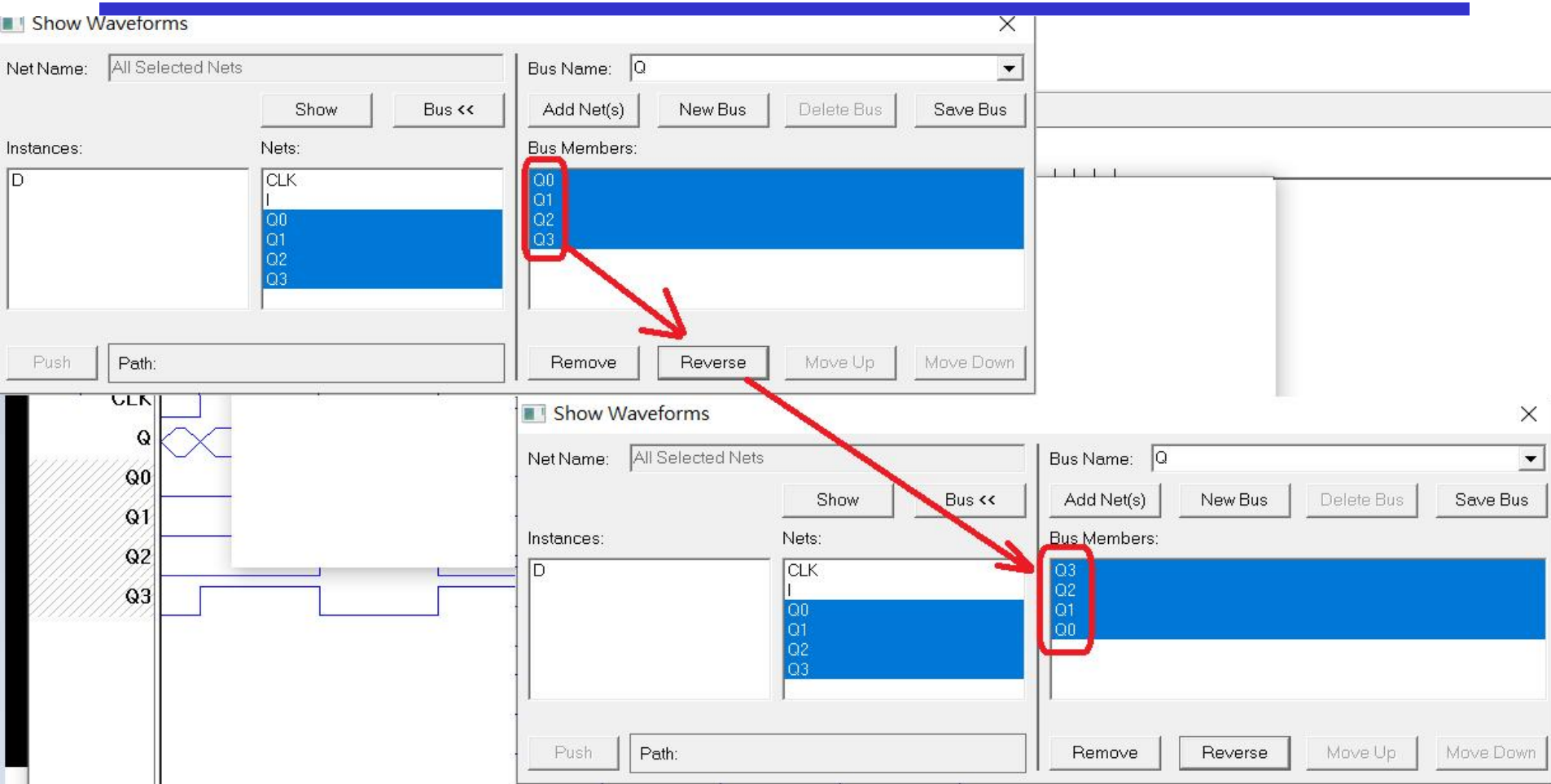
Add Net(s) New Bus Delete Bus Save Bus

Bus Members: Q3  
Q2  
Q1  
Q0 (6)

Remove Reverse (4) Move Up Move Down

Push Path:

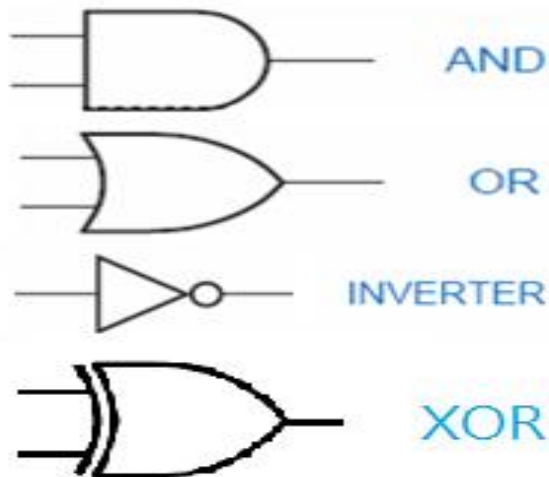
# Bit Order before and after “Reverse”



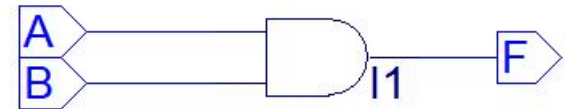
If Q3 needs to be the highest bit, but when use **Add Net(s)** button to add Q0..Q3, their order is Q0..Q3, that means Q0 is the highest bit. To correct this and make **Q3 the highest bit**, the **Reverse button** should be pressed to reverse the bit order as above.

# Simple Example --- AND Gate testing

International



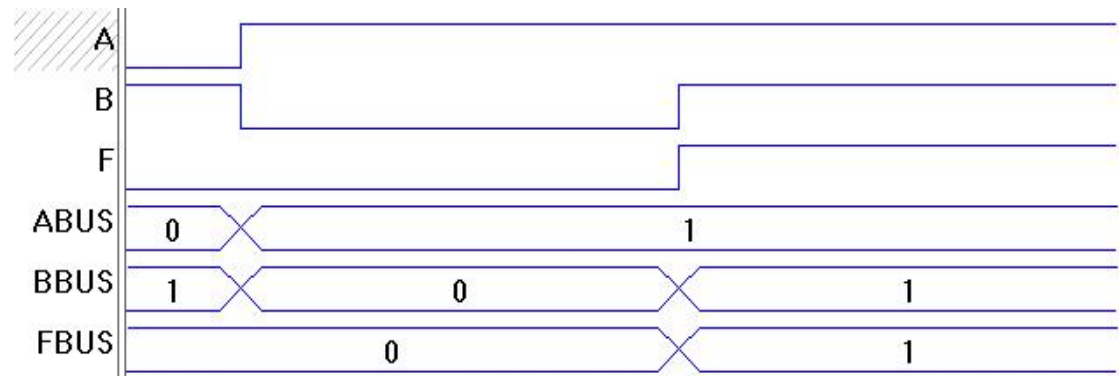
ANDGate.sch



ANDGate.abv

```
module ANDGate
A,B,F pin ;
x=.x.;
test_vectors
([A,B]->[F])
[0,0]->[x];
[0,1]->[x];
[1,0]->[x];
[1,1]->[x];
end
```

AND		
A	B	$F = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1





## Exercise 1 and 2 --- Testing OR and XOR Gates

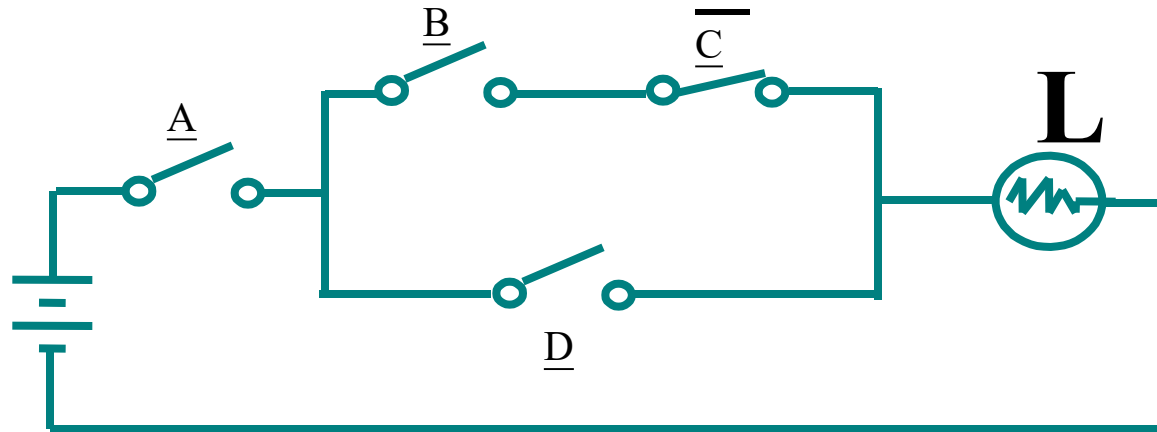
Please learning from the above testing example for **AND** gate to test **OR** and **XOR** gates, given their truth tables below:

OR		
A	B	$F = A+B$
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
A	B	$F = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

## Exercise 3 --- Logic Function Implementation by using AND, OR and Not gates forming by switches

This Logic circuit is built by using **Switches**



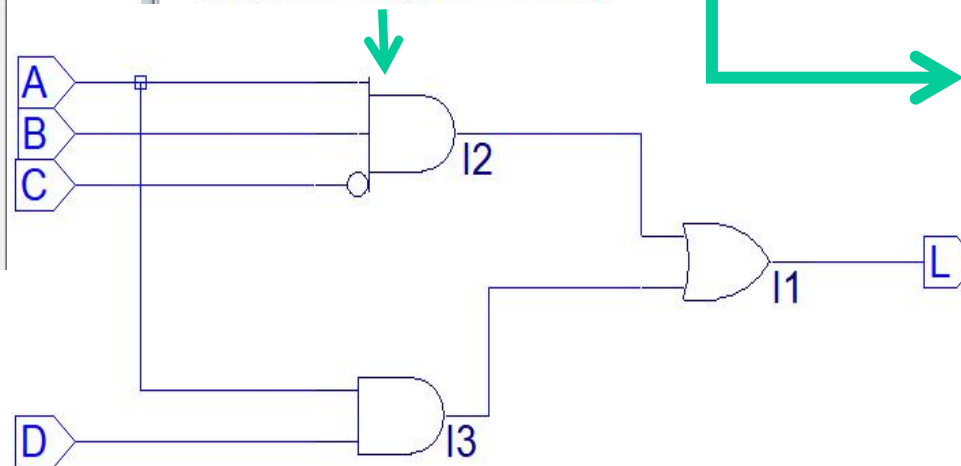
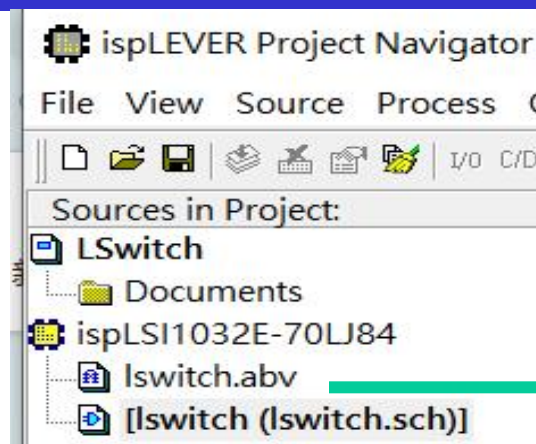
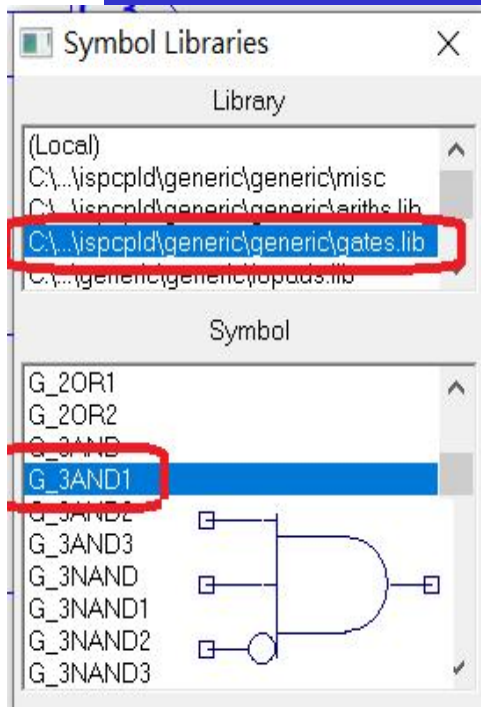
Light is on ( $L = 1$ ) for

$$L(A, B, C, D) = A ((B \overline{C}) + D) = A B \overline{C} + A D$$

and off ( $L = 0$ ), otherwise.

Please implement this circuit by **using AND, OR and Not gates with ispExpert.**

# Solution for Exercise 3

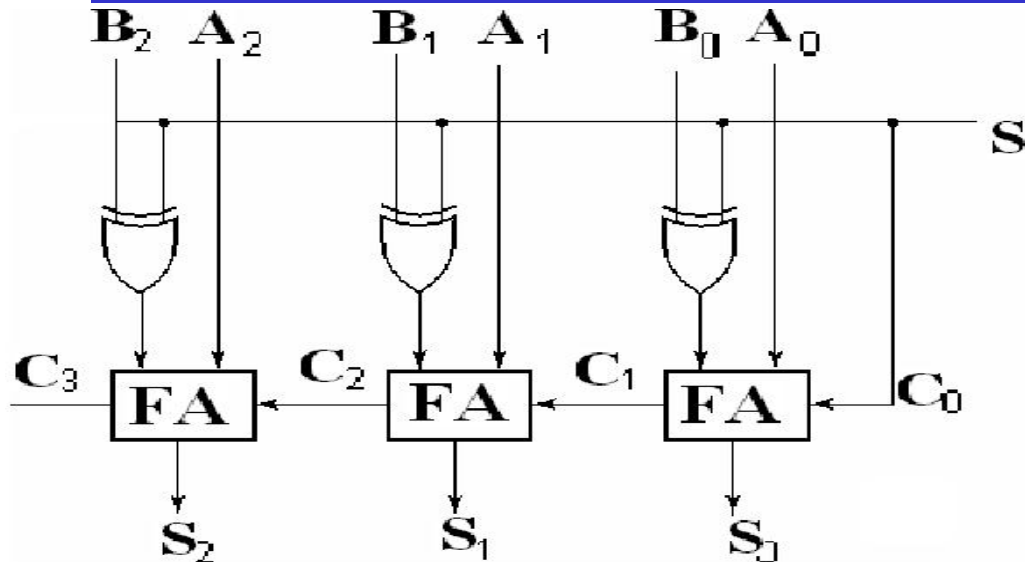


```

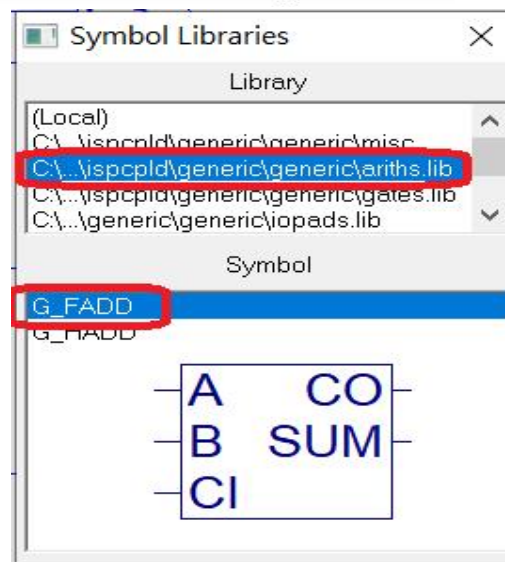
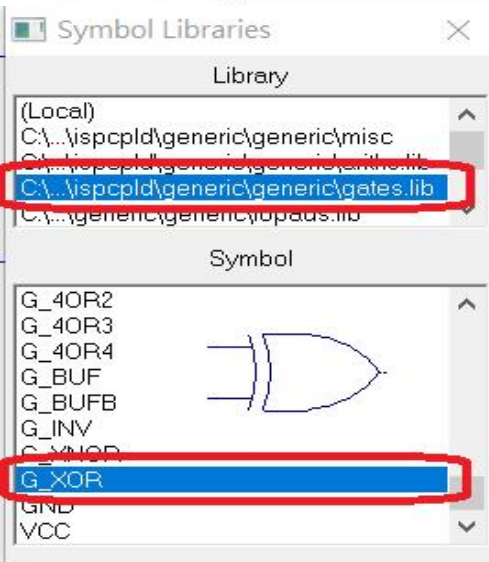
module LSwitch
A,B,C,D,L pin;
ABCD=[A,B,C,D];
x=.x.;
test_vectors
([ABCD]->[L])
[0]->[x];
[1]->[x];
[2]->[x];
[3]->[x];
[4]->[x];
[5]->[x];
[6]->[x];
[7]->[x];
[8]->[x];
[9]->[x];
[10]->[x];
[11]->[x];
[12]->[x];
[13]->[x];
[14]->[x];
[15]->[x];
end
    
```

ABCD	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LBUS	0								1	0	1		0		1	

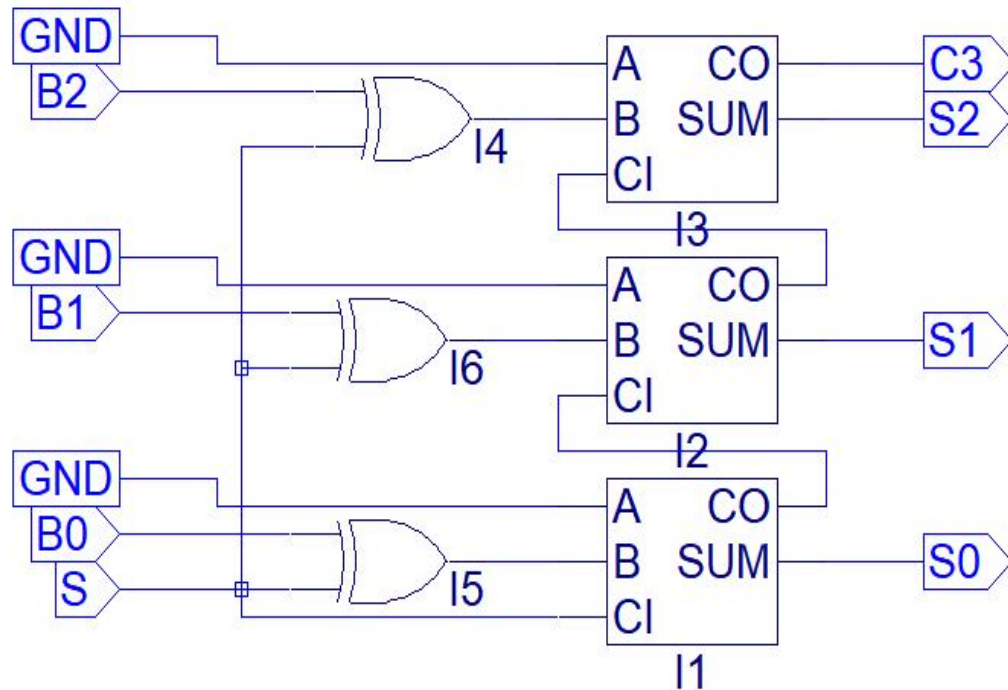
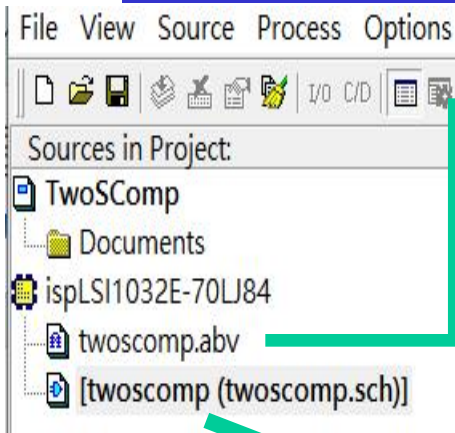
# Exercise 4 --- Implement the circuit for 3-bit 2's complement



2's complement					
B2	B1	B0	S2	S1	S0
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	0	1	1
1	1	0	0	1	0
1	1	1	0	0	1



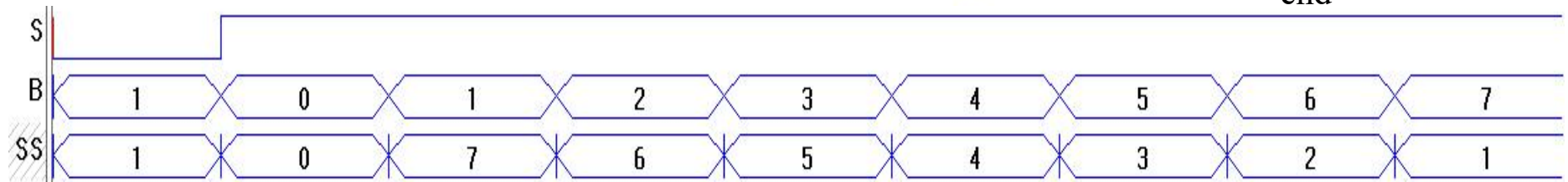
# Solution for Exercise 4



```

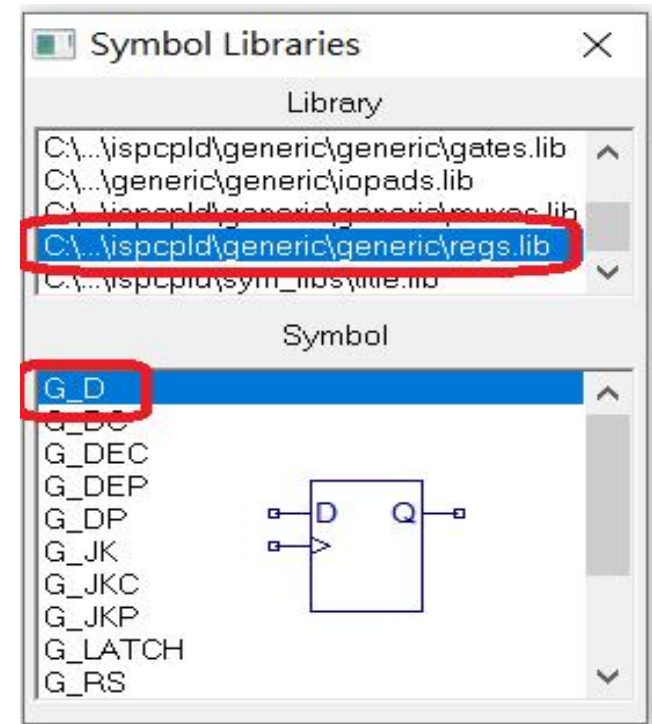
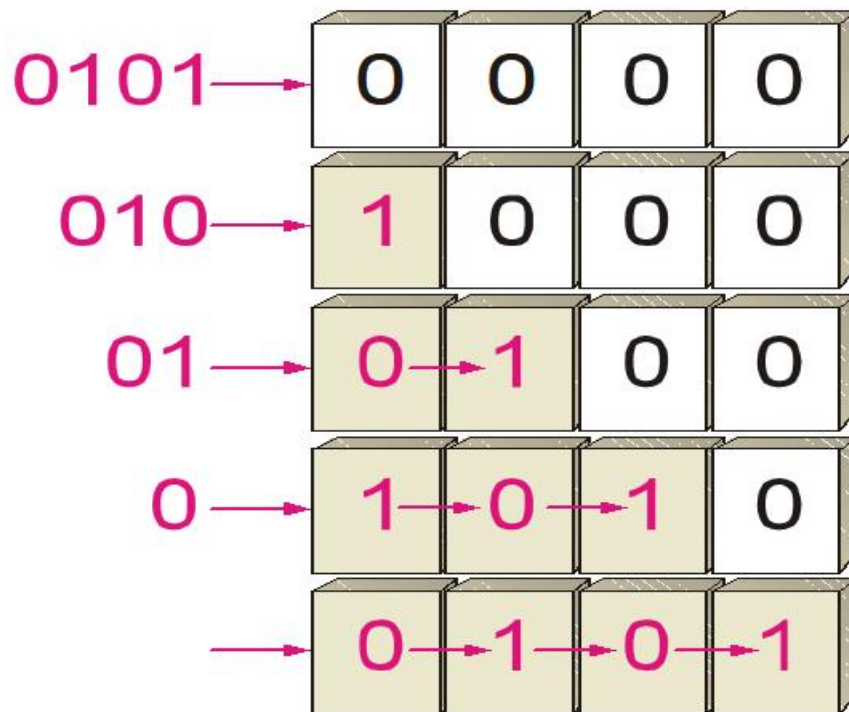
module TwoSComp
  B2..B0,S,S2..S0 pin;
  x=.x. ;
  B=[B2..B0];
  SS=[S2..S0];
  test_vectors
    ([S,B]->[SS])
    [0,1]->[x];
    [1,0]->[x];
    [1,1]->[x];
    [1,2]->[x];
    [1,3]->[x];
    [1,4]->[x];
    [1,5]->[x];
    [1,6]->[x];
    [1,7]->[x];
  end

```



# Exercise 5 --- Implement the circuit for 4-bit Shift Register

A 4-bit shift register can store a 4-bit value after **4 clock** cycles, and it can be fetched after another 4 clock cycles. Each bit can be formed by a **D flip-flop**.





# Solution for Exercise 5

